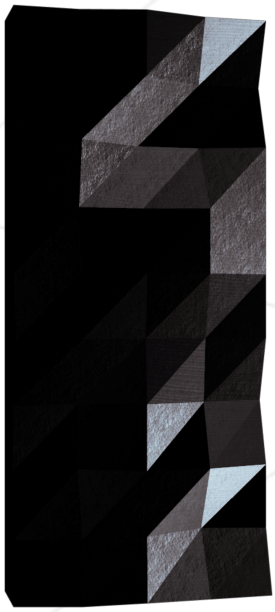

Minishell

As beautiful as a shell

Staff pedago pedago@42.fr

Summary: The objective of this project is for you to create the simplest start of a shell script. Shell is beautiful! Isn't there a famous saying? "As beautiful as Shell?" Thanks to all the shell projects, you will connect with the infinite power of Mankind Intelligence (Not even sure that you deserve it, but I have been forced to do so, so here we are...).

Contents

Chapter I

Foreword

I.1 The coder. The modern Samurai!

Actually, I have no clue about the times of the Samurai, so perhaps this comparison might not be totally accurate.

As we work on these shells (and there are quite a few), we will also be looking at the basics of Kendo. Indeed, knowing about Kendo is important to practice your programming skills.

Kendo is a modern Japanese martial art, which descended from swordsmanship. The basic idea is that when you lose, You DIE [1] Therefore it is highly recommended that you train as hard as humanly possible to stand a chance to survive and most importantly to understand some basics, especially discipline.

Kendo is similar to programming in the way that we are like the Modern Samurai. We make a mistake and it's over. I don't need to remind you that nowadays, surgery does not happen without using technology and we are the ones responsible to code that system.

So let's start with the notorious crash of Ariane 5 in 1996. It all happened because of ONE bit (not a byte. I am really talking about 1 bit).

[1] And everyone knows that getting killed is a total drag.

I.2 Ariane 5 Flight 501, or "In praise of a very well made program"

Below are bits and pieces of the investigation report. I have preffered to copy paste it for you as clicking on the link could be confusing.

You're gonna fail!

Ariane 5's first test flight (Ariane 5 Flight 501) on 4 June 1996 failed. The rocket self-destructed 37 seconds after launch because of a malfunction in the control software, which was arguably one of the most expensive computer bugs in history. A data conversion from 64-bit floating point value to 16-bit signed integer value to be stored in a variable representing horizontal bias caused a processor trap (operand error) because the floating point value was too large to be represented by a 16-bit signed integer. The software was originally written for the Ariane 4 where efficiency considerations (the computer running the software had an 80% maximum workload requirement) led to 4 variables being protected with a handler while 3 others, including the horizontal bias variable, were left unprotected because it was thought that they were "physically limited or that there was a large margin of error". The software, written in Ada, was included in the Ariane 5 through the reuse of an entire Ariane 4 subsystem despite the fact that the particular software containing the bug, which was just a part of the subsystem, was not required by the Ariane 5 because it has a different preparation sequence than the Ariane 4.

I.2.1 Analysis of the failure

In general terms, the Flight Control System of the Ariane 5 is of a standard design. The attitude of the launcher and its movements in space are measured by an Inertial Reference System (SRI). It has its own internal computer, in which angles and velocities are calculated on the basis of information from a "strap-down" inertial platform, with laser gyros and accelerometers. The data from the SRI are transmitted through the databus to the On-Board Computer (OBC), which executes the flight program and controls the nozzles of the solid boosters and the Vulcain cryogenic engine, via servovalves and hydraulic actuators. In order to improve reliability there is considerable redundancy at equipment level. There are two SRIs operating in parallel, with identical hardware and software. One SRI is active and one is in "hot" stand-by, and if the OBC detects that the active SRI has failed it immediately switches to the other one, provided that this unit is functioning properly. Likewise there are two OBCs, and a number of other units in the Flight Control System are also duplicated. The design of the Ariane 5 SRI is practically the same as that of an SRI which is presently used on Ariane 4, particularly as regards the software. Based on the extensive documentation and data on the Ariane 501 failure made available to the Board, the following chain of events, their inter-relations and causes have been established, starting with the destruction of the launcher and tracing back in time towards the primary cause.

I.2.2 Chain of technical events

- The launcher started to disintegrate at about H0 + 39 seconds because of high aerodynamic loads due to an angle of attack of more than 20 degrees that led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher.

- This angle of attack was caused by full nozzle deflections of the solid boosters and the Vulcain main engine.

- These nozzle deflections were commanded by the On-Board Computer (OBC) software on the basis of data transmitted by the active Inertial Reference System (SRI 2). Part of these data at that time did not contain proper flight data, but showed a diagnostic bit pattern of the computer of the SRI 2, which was interpreted as flight data.

- The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception.

- The OBC could not switch to the back-up SRI 1 because that unit had already ceased to function during the previous data cycle (72 milliseconds period) for the same reason as SRI 2.

- The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.

- The error occurred in a part of the software that only performs alignment of the strap-down inertial platform. This software module computes meaningful results only before lift-off. As soon as the launcher lifts off, this function serves no purpose.

- The alignment function is operative for 50 seconds after starting of the Flight Mode of the SRIs which occurs at H0 - 3 seconds for Ariane 5. Consequently, when lift-off occurs, the function continues for approx. 40 seconds of flight. This time sequence is based on a requirement of Ariane 4 and is not required for Ariane 5.

- The Operand Error occurred due to an unexpected high value of an internal alignment function result called BH, Horizontal Bias, related to the horizontal velocity sensed by the platform. This value is calculated as an indicator for alignment precision over time.

- The value of BH was much higher than expected because the early part of the trajectory of Ariane 5 differs from that of Ariane 4 and results in considerably higher horizontal velocity values.

I.2.3 Conclusion

The Board reached the following findings:

- During the launch preparation campaign and the count-down no events occurred which were related to the failure.
- The meteorological conditions at the time of the launch were acceptable and did not play any part in the failure. No other external factors have been found to be of relevance.
- Engine ignition and lift-off were essentially nominal and the environmental effects (noise and vibration) on the launcher and the payload were not found to be relevant to the failure. Propulsion performance was within specification.
- 22 seconds after H0 (command for main cryogenic engine ignition), variations of 10 Hz frequency started to appear in the hydraulic pressure of the actuators which control the nozzle of the main engine. This phenomenon is significant and has not yet been fully explained, but after consideration it has not been found relevant to the failure.
- At 36.7 seconds after H0 (approx. 30 seconds after lift-off) the computer within the back-up inertial reference system, which was working on stand-by for guidance and attitude control, became inoperative. This was caused by an internal variable related to the horizontal velocity of the launcher exceeding a limit which existed in the software of this computer.
- Approx. 0.05 seconds later the active inertial reference system, identical to the back-up system in hardware and software, failed for the same reason. Since the back-up inertial system was already inoperative, correct guidance and attitude information could no longer be obtained and loss of the mission was inevitable.
- As a result of its failure, the active inertial reference system transmitted essentially diagnostic information to the launcher's main computer, where it was interpreted as flight data and used for flight control calculations.
- On the basis of those calculations the main computer commanded the booster nozzles, and somewhat later the main engine nozzle also, to make a large correction for an attitude deviation that had not occurred.
- A rapid change of attitude occurred which caused the launcher to disintegrate at 39 seconds after H0 due to aerodynamic forces.
- Destruction was automatically initiated upon disintegration, as designed, at an altitude of 4 km and a distance of 1 km from the launch pad.
- The debris was spread over an area of 5 x 2.5 km2. Amongst the equipment recovered were the two inertial reference systems. They have been used for analysis.
- The post-flight analysis of telemetry data has listed a number of additional anomalies which are being investigated but are not considered significant to the failure.

- The inertial reference system of Ariane 5 is essentially common to a system which is presently flying on Ariane 4. The part of the software which caused the interruption in the inertial system computers is used before launch to align the inertial reference system and, in Ariane 4, also to enable a rapid realignment of the system in case of a late hold in the countdown. This realignment function, which does not serve any purpose on Ariane 5, was nevertheless retained for commonality reasons and allowed, as in Ariane 4, to operate for approx. 40 seconds after lift-off.

- During design of the software of the inertial reference system used for Ariane 4 and Ariane 5, a decision was taken that it was not necessary to protect the inertial system computer from being made inoperative by an excessive value of the variable related to the horizontal velocity, a protection which was provided for several other variables of the alignment software. When taking this design decision, it was not analysed or fully understood which values this particular variable might assume when the alignment software was allowed to operate after lift-off.

- In Ariane 4 flights using the same type of inertial reference system there has been no such failure because the trajectory during the first 40 seconds of flight is such that the particular variable related to horizontal velocity cannot reach, with an adequate operational margin, a value beyond the limit present in the software.

- Ariane 5 has a high initial acceleration and a trajectory which leads to a build-up of horizontal velocity which is five times more rapid than for Ariane 4. The higher horizontal velocity of Ariane 5 generated, within the 40-second timeframe, the excessive value which caused the inertial system computers to cease operation.

- The purpose of the review process, which involves all major partners in the Ariane 5 programme, is to validate design decisions and to obtain flight qualification. In this process, the limitations of the alignment software were not fully analysed and the possible implications of allowing it to continue to function during flight were not realised.

- The specification of the inertial reference system and the tests performed at equipment level did not specifically include the Ariane 5 trajectory data. Consequently the realignment function was not tested under simulated Ariane 5 flight conditions, and the design error was not discovered.

- It would have been technically feasible to include almost the entire inertial reference system in the overall system simulations which were performed. For a number of reasons it was decided to use the simulated output of the inertial reference system, not the system itself or its detailed simulation. Had the system been included, the failure could have been detected.

- Post-flight simulations have been carried out on a computer with software of the inertial reference system and with a simulated environment, including the actual trajectory data from the Ariane 501 flight. These simulations have faithfully reproduced the chain of events leading to the failure of the inertial reference systems.

The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds

after lift- off). This loss of information was due to specification and design errors in the software of the inertial reference system.

The extensive reviews and tests carried out during the Ariane 5 Development Programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure.

`At least try to read this...`

Everything came down to one tiny variable: the one allocated to horizontal acceleration. Actually, the maximum acceleration of Ariane 4 was about 64, the variable was coded to 8 bits. In a computer, the information is coded in a rather special alphabet called binary language. A bit is equivalent to a letter in the alphabet containing both letters “0” and “1”; so any word (or variable value) is written by a combination of 8 letters, each of those letters being either “0” or “1”. In binary base, that means $28 = 256$ possible values (256 combinations of 8 bits), enough to code the value 64 written 1000000 and needing 8 bits. But Ariane 5 was faster: its acceleration could reach the value 300 (worth 100101100 in binary and needing 9 bits). Therefore, the coded variable on 8 bits had a capacity limitation since its memory space wasn’t large enough to accept such a high value, it would have to have been code with one more bit, ie 9, which would have given $29 = 512$ as a highest value, therefore enough to code the 300 value. From this over capacity resulted an absurd value in the variable that did not match the reality. With a domino effect, the program decided to self-destruct the rocket from that one data error.

I.3 A functioning link

The complete investigation report

I.4 How to facilitate the completion of this project.

`If did not read the foreword, you are a moron.`

This project will be easier to complete if you can admit to yourself once and for all that you can avoid the crash if you run 3 simple tests as you go along!

Chapter II

Introduction

The existence of shells is linked to the very existence of IT. At the time, all coders agreed that `communicating with a computer using aligned 1/0 switches was seriously irritating`. It was only logical that they came up with the idea `to communicate with a computer using an interactive lines of commands in a language somewhat close to english`.

With `Minishell`, you'll be able to travel through time and come back to problems people faced when `Windows` didn't exist. If this doesn't make you a better coder, nothing will.

Chapter III

Objectives

Through the `Minishell` project, you will get to the core of the `Unix` system and explore an important part of this system's API: process creation and synchronisation. Executing a command inside a shell implies creating a new process, which execution and final state will be monitored by its parent's process. This set of functions will be the key to success for your `Minishell`, so be sure to code the cleanest, simplest program possible. Otherwise, you'll probably have to start from scratch for your `21sh` project and that would be a real shame.

Be rigorous and methodical in your `C` coding, take the necessary time to read and understand the `mans`, but most importantly, test your code!

Chapter IV

General Instructions

- This project will only be corrected by actual human beings. You are therefore free to organize and name your files as you wish, although you need to respect some requirements listed below.
- The executable file must be named `minishell`.
- You must submit a `Makefile`. That `Makefile` needs to compile the project and must contain the usual rules. It can only recompile the program if necessary.
- If you are clever, you will use your library for your `minishell`. Also submit your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- Your project must be written in `C` in accordance with the Norm.
- You have to handle errors in a sensitive manner. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Your program cannot have memory leaks.
- You'll have to submit at the root of your folder, a file called `author` containing your username followed by a '\n'

```
$>cat -e author
xlogin$
```

- Within your mandatory part you are allowed to use the following functions:
  - `malloc`, `free`
  - `access`
  - `open`, `close`, `read`, `write`
  - `opendir`, `readdir`, `closedir`
  - `getcwd`, `chdir`
  - `stat`, `lstat`, `fstat`

- `fork`, `execve`
- `wait`, `waitpid`, `wait3`, `wait4`
- `signal`, `kill`
- `exit`

- You are allowed to use other functions to carry out the bonus part as long as their use is justified during your defence. For example, to use `tcgetattr` is justified in certain case, to use `printf` because you are lazy isn't. Be smart!

- You can ask questions on the forum & Slack.

Chapter V

Mandatory part

- You must program a mini `UNIX` command interpreter.
- This interpreter must display a prompt (a simple "`$>` " for example) and wait till you type a command line, validated by pressing enter.
- The prompt is shown again only once the command has been completely executed.
- The command lines are simple, no pipes, no redirections or any other advanced functions.
- The executable are those you can find in the paths indicated in the `PATH` variable.
- In cases where the executable cannot be found, it has to show an error message and display the prompt again.
- You must manage the errors without using `errno`, by displaying a message adapted to the error output.
- You must deal correctly with the `PATH` and the environment (copy of system `char **environ`).
- You must implement a series of builtins: `echo`, `cd`, `setenv`, `unsetenv`, `env`, `exit`.
- You can choose as a reference whatever shell you prefer.
- You must manage expansions $ and ~

`Read the man carefully.`

Here is a use example of your `minishell` :

```
$> cd /dev
$> pwd
/dev
$> ls -l
total 0
crw-rw----  1 root   video     10, 175 dec 19 09:50 agpgart
lrwxrwxrwx  1 root   root            3 dec 19 09:50 cdrom -> hdc
lrwxrwxrwx  1 root   root            3 dec 19 09:50 cdrom0 -> hdc
drwxr-xr-x  2 root   root           60 dec 19 09:50 cdroms/
lrwxrwxrwx  1 root   root            3 dec 19 09:50 cdrw -> hdc
lrwxrwxrwx  1 root   root           11 dec 19 09:50 core -> /proc/kcore
drwxr-xr-x  3 root   root           60 dec 19 09:50 cpu/
drwxr-xr-x  3 root   root           60 dec 19 09:50 discs/
lrwxrwxrwx  1 root   root            3 dec 19 09:50 disk -> hda
lrwxrwxrwx  1 root   root            3 dec 19 09:50 dvd -> hdc
lrwxrwxrwx  1 root   root            3 dec 19 09:50 dvdrw -> hdc
crw-------  1 root   root      29,   0 dec 19 09:50 fb0
lrwxrwxrwx  1 root   root           13 dec 19 09:50 fd -> /proc/self/fd/
brw-rw----  1 root   floppy     2,   0 dec 19 09:50 fd0
brw-rw----  1 root   floppy     2,   1 dec 19 09:50 fd1
crw-rw-rw-  1 root   root       1,   7 dec 19 09:50 full
brw-rw----  1 root   root       3,   0 dec 19 09:50 hda
brw-rw----  1 root   root       3,   1 dec 19 09:50 hda1
brw-rw----  1 root   root       3,   2 dec 19 09:50 hda2
brw-rw----  1 root   root       3,   3 dec 19 09:50 hda3
brw-rw----  1 root   root       3,   5 dec 19 09:50 hda5
brw-rw----  1 root   root       3,   6 dec 19 09:50 hda6
$> kwame
kwame: command not found
$>
```

Chapter VI

Bonus part

Quite a few features will be on the menu of the 21sh and 42sh projects. Below are some bonuses that you can start implementing immediately. Only if you wish to do so!

- Management of signals and in particular `Ctrl-C`. The use of global variables is allowed as part of this bonus.
- Management of execution rights in the PATH.
- Auto completion.
- The separation of commands with ";".
- Other bonuses that you will think to be useful.

Chapter VII

Submission and peer correction

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.

Good luck to all! And remember to push your author file!